STAGGER: Periodicity Mining of Data Streams using Expanding Sliding Windows *

Mohamed G. Elfeky[1] Walid G. Aref[2] Ahmed K. Elmagarmid[2]

[1]Google Inc., Mountain View, California

[2]Department of Computer Sciences, Purdue University
mgelfeky@google.com, {aref, ake}@cs.purdue.edu

Abstract

Sensor devices are becoming ubiquitous, especially in measurement and monitoring applications. Because of the real-time, append-only and semi-infinite natures of the generated sensor data streams, an online incremental approach is a necessity for mining stream data types. In this paper, we propose STAGGER: a one-pass, online and incremental algorithm for mining periodic patterns in data streams. STAGGER does not require that the user pre-specify the periodicity rate of the data. Instead, STAGGER discovers the potential periodicity rates. STAGGER maintains multiple expanding sliding windows staggered over the stream, where computations are shared among the multiple overlapping windows. Small-length sliding windows are imperative for early and real-time output, yet are limited to discover short periodicity rates. As streamed data arrives continuously, the sliding windows expand in length in order to cover the whole stream. Larger-length sliding windows are able to discover longer periodicity rates. STAGGER incrementally maintains a tree-like data structure for the

*This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

frequent periodic patterns of each discovered potential periodicity rate. In contrast to the Fourier/Wavelet-based approaches used for discovering periodicity rates, STAGGER not only discovers a wider, more accurate set of periodicities, but also discovers the periodic patterns themselves. In fact, experimental results with real and synthetic data sets show that STAGGER outperforms Fourier/Wavelet-based approaches by an order of magnitude in terms of the accuracy of the discovered periodicity rates. Moreover, real-data experiments demonstrate the practicality of the discovered periodic patterns.

Keywords: Mining data streams, Periodicity mining

1 Introduction

Data streams consist of continuous and time sensitive data. Periodicity mining is a tool that helps in predicting the behavior of data streams. For example, periodicity mining allows a telephone company to analyze telephone calling patterns and predict periods of high and low usage so that proper planning may take place. In this paper, we address the problem of mining such periodic patterns over data streams. We define periodicity mining as the detection of frequent periodic patterns where the periodicity rate (period length) is also unknown. Therefore, the first step, termed *Periodicity Detection*, is to discover potential periodicity rates. The second step, termed *Mining Periodic Patterns*, is to detect the frequent periodic patterns of each discovered periodicity rate. While this two-step process works for traditional time series data [32, 25], it does not apply well to real-time data streams. Furthermore, data streams are potentially infinite and therefore require online and incremental processing. Recently, Papadimitriou et al. [29] have addressed the problem of periodicity detection (the first step, above) in data streams using Wavelets. In this paper, we propose a complete solution to the periodicity mining problem in data streams, which combines both periodicity detection and mining periodic patterns in a one-pass algorithm. Moreover, in contrast to the techniques proposed in this paper, the Wavelet-based approaches are approximate. The experiments demonstrate that the quality of the proposed techniques are superior in terms of the accuracy of the discovered periodicity rates.

In this paper, we propose to maintain multiple expanding sliding windows that are staggered over the data stream. Using a convolution-based algorithm [14, 15], a sliding window[1] of length w can discover period lengths up to $w/2$. Whereas a small window length is required for early and real-time output, it limits the period lengths that can be discovered. Therefore, we propose STAGGER, a new algorithm that uses the algorithm in [14] as a building block. STAGGER allows multiple small sliding windows to expand in length to cover the whole data stream, i.e., STAGGER *staggers* the stream with multiple and concurrent expanding sliding windows. Computations are shared among the multiple overlapping windows. Thus, STAGGER is able to produce interactive output as well as to discover a wide range of potential period lengths. Subsequently, we maintain a max-subpattern tree [19] for each potential period length, which discovers the periodic patterns for that length. We propose an approximate incremental technique for building and maintaining max-subpattern trees.

A related problem to online mining of data streams is that of oscillating patterns. Traditional data mining techniques employ a single frequency threshold value. A pattern is reported as frequent only if its frequency is above that threshold. When data arrives continuously, a pattern may oscillate between being frequent and being infrequent. Maintaining a data structure for frequent patterns, e.g., the max-subpattern tree, such an oscillating pattern loses its history information every time it becomes infrequent. When it becomes frequent again, the pattern's history has already gone without being stored. Losing a pattern's history deteriorates the accuracy of one-pass online stream mining algorithms. A naïve solution is to keep all patterns' histories even for those patterns that become infrequent. However, this solution requires an excessive amount of storage. We propose a new approach, termed *"hysteresis" thresholding*, that maintains two thresholds. A pattern is considered frequent if its frequency is above the higher threshold value, and is considered infrequent if its frequency is below the lower threshold value. The distance between the two threshold values acts as a period of time during which a pattern is given a chance to stabilize rather than to oscillate. As we demonstrate on the experimental section, the proposed "hysteresis" thresholding approach shows significant performance improvement over that of the traditional thresholding

[1] A window is defined in terms of number of symbols rather than a time window

approach.

The contributions of this paper can be summarized as follows:

1. We propose STAGGER, an online incremental algorithm that uses expanding sliding-windows in order to discover potential periodicity rates in data streams. STAGGER uses and shares the execution among multiple expanding sliding-windows that are staggered over the data stream, in order to produce interactive output. This way, STAGGER discovers a wide range of potential periodicity rates.

2. We propose a new incremental technique for maintaining the max-subpattern tree that is used for mining periodic patterns in data streams. The proposed technique requires only one pass over the data stream and no reprocessing of data that has been previously seen.

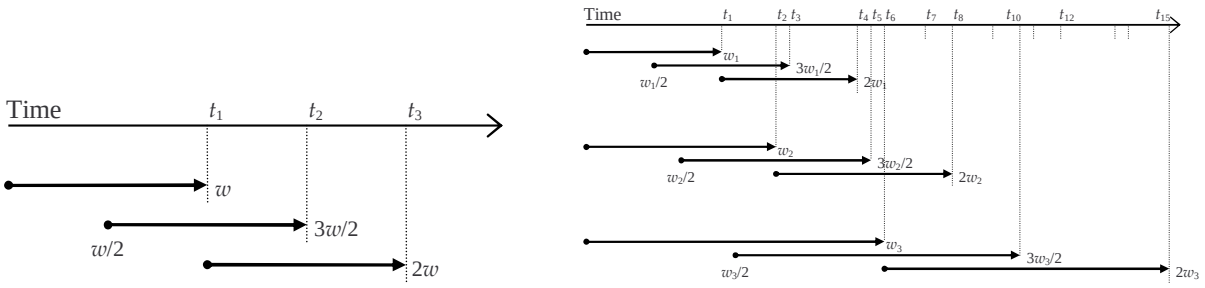
3. We propose using "hysteresis" thresholding for maintaining the mining thresholds. This approach is very relevant to one-pass online stream mining algorithms and is conservative in terms of preserving the history of candidate frequent patterns.

The rest of this paper is organized as follows. In the remaining of this section, we introduce the notation used throughout the paper. We outline STAGGER in Section 2. In Section 3, we present the proposed technique for online periodicity detection. In Section 4, we present the proposed technique for incremental mining of periodic patterns. In Section 5, we evaluate the performance of STAGGER, and compare it to other approaches using both synthetic and real data. A discussion of the related work is given in Section 6, and we conclude the paper in Section 7.

1.1 Notation

A data stream of events is an infinite sequence of timestamped events drawn from a finite set of nominal[2] event types. An example is an event stream in a computer network that monitors the various events. Let e_i be the event occurring at timestamp i, then the data stream S is represented as $S = e_0, e_1, \ldots, e_i, \ldots$. Each event type can be denoted by a symbol (e.g., a,

[2]Nominal values are distinguished by name only, and do not have an inherent order or a distance measure.



(a) A window of length w slides $w/2$ positions

(b) Three sliding windows of lengths w_1, w_2 and w_3

Figure 1: Expanding sliding windows

b, c). The set of event types can be denoted $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \cdots\}$. Thus, the data stream S can be considered a sequence of infinite length over a finite alphabet Σ.

A data stream may also be an infinite sequence of timestamped values collected by a sensor measuring a specific feature. For example, the feature in a data stream for stock prices might be the final daily stock price of a specific company. If we discretize[3] the data stream feature values into nominal discrete levels and denote each level (e.g., high, medium, low) by a symbol (e.g., a, b, c), then we can use the same notation above.

2 Algorithm Outline

In this section, we outline STAGGER algorithmic behavior. The input is a data stream for which we output its periodic patterns. We are given a periodicity threshold (or two periodicity thresholds if the "hysteresis" thresholding approach is used). Arbitrarily, we select m different windows of lengths $w_1 < w_2 < \ldots < w_m$. Notice that w_m can be selected as large as the buffer size that is allocated for buffering the data stream. STAGGER maintains an event-based priority queue where an event is triggered by the arrival of new data in the stream. The events in the priority queue are sorted in increasing order of their timestamps.

STAGGER is outlined in the following steps:

- Initialize the priority queue by the events that correspond to the arrival of a number

[3]The problem of discretizing time series into a finite alphabet is orthogonal to our problem and is beyond the scope of this paper. See [12, 22] for an exhaustive overview of discretization and segmentation techniques, and [24] for streaming implications of discretization techniques.

of symbols equal to $w_1, w_2, \ldots, w_m$ (e.g., events at Times t_1, t_2, and t_6 of Figure 1(b)).

- Get the next event from the priority queue.
- If the event corresponds to considering a new window w_i (e.g., events at Times t_1, t_2, and t_6 of Figure 1(b)), then apply a convolution-based algorithm [14] over the window w_i (Section 3.2), sharing the results from the previous window w_{i-1} (Section 3.3.2).
- If the event corresponds to sliding the window w_i (e.g., events at Times t_3, t_5, and t_{10} of Figure 1(b)), then update the results of this window (Section 3.3.1).
- Analyze the results from the previous step (Section 3.2) to obtain potential period lengths p_j, and their corresponding maximal periodic patterns.
- For each p_j, build a max-subpattern tree [19] R_{p_j} using p_j's corresponding maximal periodic pattern (Section 4).
- Schedule the next event that corresponds to sliding the current window w_i a number of positions equal to $w_i/2$, and insert that event in the priority queue (e.g., in Figure 1(b), the event at Time t_5 schedules the event at Time t_8).

3 Periodicity Detection

Periodicity detection, in our terms, stands for discovering potential rates at which the data stream is periodic. For example, a data stream of the closing price of a specific stock may have a periodicity rate of 7 that describes its weekly periodic pattern. A periodic pattern describes the periodic behavior at, not necessarily all, the points in the period. For example, the closing price of a specific stock may be *high* every Friday and *low* every Tuesday, yet may not have any regularity on the other week days. That description of periodic patterns implies that the technique devised for periodicity detection should consider symbol periodicities. Recall that a symbol may represent an event in an event data stream or a nominal discrete level in a discretized real-valued data stream.

3.1 Symbol Periodicity

In a data stream S, a symbol s is said to be periodic with a period of length p if s exists "almost" every p timestamps. For example, in the data stream $S = \mathsf{abcabbabdb}$, the symbol b is periodic with a period of length 4 since b exists every four timestamps (positions 1, 5 and 9). Moreover, the symbol a is periodic with a period of length 3 since a exists almost every three timestamps (positions 0, 3, and 6 but not 9). Let $\pi_{p,l}(S)$ denote the projection of a data stream S according to a period p starting from position l:

$$\pi_{p,l}(S) = e_l, e_{l+p}, e_{l+2p}, \ldots, e_{l+(m-1)p},$$

where $0 \leq l < p$, $m = \lceil (n-l)/p \rceil$, and n is the length of S. For example, if $S = \mathsf{abcabbabdb}$, then $\pi_{4,1}(S) = \mathsf{bbb}$, and $\pi_{3,0}(S) = \mathsf{aaab}$. Intuitively, the ratio of the number of occurrences of a symbol s in a certain projection $\pi_{p,l}(S)$ to the length of this projection indicates how often this symbol occurs every p timestamps. This ratio, however, is not quite accurate since it captures all the occurrences including the outliers. In the example above, $\pi_{3,0}(S) = \mathsf{aaab}$ implies that the symbol b is periodic with a period of length 3 with a frequency of 1/4, which is not quite true. As another example, if for a certain S, $\pi_{p,l}(S) = \mathsf{abcbac}$, then the symbol changes every p timestamps and hence no symbol should be considered periodic with a period of length p. We remedy this problem by considering only the consecutive occurrences. A consecutive occurrence of a symbol s in a certain projection $\pi_{p,l}(S)$ indicates that the symbol s has reappeared in S after p timestamps from the previous appearance. Let $\mathcal{F}_2(s, S)$ denote the number of times the symbol s occurs in two consecutive positions in the data stream S. For example, if $S = \mathsf{abbaaabaa}$, then $\mathcal{F}_2(\mathsf{a}, S) = 3$ and $\mathcal{F}_2(\mathsf{b}, S) = 1$. Therefore, the ratio of the number of consecutive occurrences of a symbol s in a certain projection $\pi_{p,l}(S)$ to the length of this projection $(\frac{\mathcal{F}_2(s,\pi_{p,l}(S))}{\lceil (n-l)/p \rceil - 1})$ indicates how often the symbol s occurs every p timestamps in a data stream S.

Definition 1 *If a data stream S of length n contains a symbol s such that $\exists l, p$ where $0 \leq l < p$, and $\frac{\mathcal{F}_2(s,\pi_{p,l}(S))}{\lceil (n-l)/p \rceil - 1} \geq \tau$ where $0 \leq \tau \leq 1$; then s is said to be periodic in S with a period of length p at position l with respect to periodicity threshold τ.*

For example, in the data stream $S = \mathsf{abcabbabdb}$, $\frac{\mathcal{F}_2(\mathsf{a},\pi_{3,0}(S))}{\lceil 10/3 \rceil - 1} = 2/3$, thus the symbol a is periodic with a period of length 3 at position 0 with respect to a periodicity threshold

$\tau \leq 2/3$. Similarly, the symbol b is periodic with a period of length 3 at position 1 with respect to a periodicity threshold $\tau \leq 1$.

The main advantage of Definition 1 is that not only does it determine the candidate periodic symbols and their corresponding periods, but also it locates their corresponding positions. In other words, each symbol that exhibits periodicity according to Definition 1 produces a frequent single-symbol periodic pattern.

Definition 2 *In a data stream S of a finite alphabet Σ, a pattern of length p is a sequence $q = q_0 \cdots q_{p-1}$, such that $q_i \subseteq \Sigma$.*

Definition 3 *If a data stream S of length n contains a symbol s that is periodic with a period of length p at position l, then a frequent single-symbol periodic pattern $q = q_0 \cdots q_{p-1}$ of length p is formed by setting all q_i's to the empty set except for q_l that is set to $\{s\}$.*

For example, in the data stream $S =$ abcabbabdb, for a periodicity threshold that is less than 2/3, the pattern[4] a$**$ is a frequent single-symbol periodic pattern of length 3, and so is the single-symbol periodic pattern $*$b$*$.

As will be described in Section 4, frequent single-symbol periodic patterns are used to form a maximal periodic pattern that is the root node of the max-subpattern tree of the corresponding period length.

Given these definitions, in the next sections, we discuss how STAGGER detects all the potential period lengths and all the corresponding frequent single-symbol periodic patterns in one pass over the data stream. At the core of STAGGER is a symbol periodicity detection algorithm [14], called SPD, that deals with data stream portions (windows) of known lengths. We present the SPD algorithm (Section 3.2) partly to render this paper self-contained, but also to facilitate introducing the multiple sliding windows online incremental technique for infinite data streams (Section 3.3).

[4]For simplicity, the symbol $*$ denotes the empty set, and the set brackets are omitted when any q_i is a singleton (contains only one element). Note that $*$ is a place-holder for only one element.

3.2 The SPD Algorithm

Assume that the period length p is known for some symbols of a specific portion, say S, of a data stream. Then, the problem is reduced to mining S for the frequent single-symbol periodic patterns of period length p. In other words, the problem is to detect the symbols that are periodic with period length p within S. A way to solve this simpler problem is to shift S by p positions, denoted as $S^{(p)}$, and then compare $S^{(p)}$ to S. For example, if $S = \mathsf{abcabbabcb}$, then shifting S three positions results in $S^{(3)} = ***\mathsf{abcabba}$. Comparing S to $S^{(3)}$ results in four symbol matches. If the symbols are mapped in a particular way, we can deduce that these four matches are actually two for the symbol a and two for the symbol b.

The SPD algorithm [14] relies on two main ideas to detect symbol periodicities in fixed-length data stream portions. The first idea is to use the concept of convolution in order to shift and compare the data stream portion for all possible values at once. The second idea is to obtain a mapping scheme for the symbols, which reveals, upon comparison, the identity of which symbols that match and their corresponding positions. The remaining part of this section describes these ideas in detail. The convolution is covered in Section 3.2.1 while the mapping scheme is covered in Section 3.2.2.

3.2.1 Convolution

A convolution [11] is defined as follows. Let $X = [x_0, x_1, \ldots, x_{n-1}]$ and $Y = [y_0, y_1, \ldots, y_{n-1}]$ be two finite length sequences of numbers[5], each of length n. The convolution of X and Y is defined as another finite length sequence $X \otimes Y$ of length n such that

$$(X \otimes Y)_i = \sum_{j=0}^{i} x_j y_{i-j}$$

for $i = 0, 1, \ldots, n-1$. Let $X' = [x'_0, x'_1, \ldots, x'_{n-1}]$ denote the reverse of the vector X, i.e., $x'_i = x_{n-1-i}$. Taking the convolution of X' and Y, and obtaining its reverse leads to the following:

$$(X' \otimes Y)'_i = (X' \otimes Y)_{n-1-i} = \sum_{j=0}^{n-1-i} x'_j y_{n-1-i-j} = \sum_{j=0}^{n-1-i} x_{n-1-j} y_{n-1-i-j},$$

[5]The general definition of convolution does not assume equal-length sequences. We adapt the general definition to conform to our problem, in which convolutions only take place between equal-length sequences.

i.e.,

$$
\begin{aligned}
(X' \otimes Y)'_0 &= x_0y_0 + x_1y_1 + \cdots + x_{n-1}y_{n-1},\\
(X' \otimes Y)'_1 &= x_1y_0 + x_2y_1 + \cdots + x_{n-1}y_{n-2},\\
&\vdots\\
(X' \otimes Y)'_{n-1} &= x_{n-1}y_0.
\end{aligned}
$$

In other words, the component of the resulting sequence at position i corresponds to shifting one of the input sequences i positions and comparing it to the other input sequence.

The SPD algorithm performs the following steps: (i) Convert the data stream portion S into two finite sequences of numbers $\Phi(S)$ and $\Phi(S)'$, where $\Phi(S)'$ is the reverse of $\Phi(S)$ (based on the mapping scheme Φ described in Section 3.2.2), (ii) Perform the convolution between the two sequences $\Phi(S)' \otimes \Phi(S)$, and (iii) Reverse the output $\Big(\Phi(S)' \otimes \Phi(S)\Big)'$. The component values of the resulting sequence correspond to shifting and comparing the data stream portion for all possible values.

It is well known that convolution can be computed by the fast Fourier transform (FFT) [23] as follows:

$$X \otimes Y = \text{FFT}^{-1}\Big(\text{FFT}(X) \cdot \text{FFT}(Y)\Big).$$

This computation reduces the time complexity of the convolution to $O(n \log n)$. The brute-force approach of shifting and comparing the data stream portion for all possible values has the time complexity $O(n^2)$.

3.2.2 Mapping Scheme

Let $S = e_0, e_1, \ldots, e_{n-1}$ be a data stream portion of length n, where e_i's are symbols from a finite alphabet Σ of size σ. Let Φ be a mapping for the symbols of S such that $\Phi(S) = \Phi(e_0), \Phi(e_1), \ldots, \Phi(e_{n-1})$. Let $C(S) = \Big(\Phi(S)' \otimes \Phi(S)\Big)'$, and $c_i(S)$ be the ith component of $C(S)$. The mapping scheme of the SPD algorithm satisfies two conditions: (i) While matched symbols contribute a non-zero value in the product $\Phi(e_j) \cdot \Phi(e_{i-j})$, unmatched symbols contribute 0, and (ii) the value of each component of $C(S)$, $c_i(S) = \sum_{j=0}^{i} \Phi(e_j) \cdot \Phi(e_{i-j})$, identifies the matched symbols and their corresponding positions.

The SPD algorithm maps the symbols to the binary representation of increasing powers of 2. For example, if a data stream contains only the 3 symbols a, b, and c, then a possible mapping would be a : 001, b : 010, and c : 100, which correspond to power values of 0, 1, and 2, respectively. Hence, a data stream portion of length n is converted to a binary vector of length σn. For example, let $S = \mathsf{acccabb}$, then S is converted to the binary vector $\bar{S} = 001100100100001010010$. Using convolution results in a sequence $C(\bar{S})$ of length σn. Considering just the n positions $0, \sigma, 2\sigma, \ldots, (n-1)\sigma$ that are the exact start positions of the symbols, gives back the sequence $C(S)$. The latter derivation of $C(S)$ can be written as $C(S) = \pi_{\sigma,0}(C(\bar{S}))$ using the notation defined in Section 3.1.

The SPD algorithm modifies the definition of convolution as follows:

$$(X \otimes Y)_i = \sum_{j=0}^{i} 2^j x_j y_{i-j}.$$

Notice that this definition still preserves that

$$X \otimes Y = \text{FFT}^{-1}\Big(\text{FFT}(X) \cdot \text{FFT}(Y)\Big).$$

The reason for adding the coefficient 2^j is to get a different contribution for each match. For example, Figure 2 illustrates an example where $S = \mathsf{acccabb}$. The powers of 2 in the value of $c_1(S)$ are 1, 11, and 14. Examining those powers modulo 3, which is the size of the alphabet in this particular example, results in 1, 2 and 2, respectively, which correspond to the symbols b, c, and c, respectively.

$\bar{T}$:	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	0
$\bar{T}^{(3)}$:			0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	
$c_3(\bar{T}) = c_1(T) =$						2^{14}	+	2^{11}	+										2^1		
$\bar{T}$:	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	0
$\bar{T}^{(12)}$:													0	0	1	1	0	0	1	0	0
$c_{12}(\bar{T}) = c_4(T) =$															2^6						

Figure 2: A clarifying example for the mapping scheme

Figure 2 gives another example for $c_4(S)$ that has only one power of 2, which is 6, that corresponds to the symbol a since 6 mod 3 = 0 and a was originally mapped to the binary representation of 2^0. This means that comparing S to $S^{(4)}$ results in only one match of the symbol a. Moreover, the power value of 6 reveals that the symbol a is at position 0 in

$S^{(4)}$. Therefore, the power values reveal not only the number of matches of each symbol at each period, but also the corresponding starting positions. This latter observation complies with the definition of symbol periodicity. Hereby, we direct the reader to [14] for a complete discussion of that algorithm.

3.3 Online Symbol Periodicity Detection

The idea of STAGGER is to perform the SPD algorithm incrementally on multiple sliding windows of expanding lengths. Sliding a window implies discarding an earlier portion of the data stream and considering a more recent portion. After sliding a window, the results are carried from the previous step to the current one. The SPD algorithm is performed only over the recent portion of the data stream without losing any information about the discarded portion. Moreover, the results are shared among the multiple windows in order to avoid recomputing what has been computed earlier.

We introduce the notion of a single sliding window as the basic step of STAGGER (Section 3.3.1). Then, we justify the need for multiple sliding windows, and show how the results are shared among them (Section 3.3.2).

3.3.1 Single Sliding-Window

Let $S = e_0, e_1, \ldots$ be an infinite length data stream, where e_i's are symbols from a finite alphabet Σ of size σ. Consider a window of length w, and let $S_{i,w}$ denote the portion of the data stream S that is of length w and starts at position i. Performing the SPD algorithm over the very first window $S_{0,w}$ discovers all potential periods of values that range from 1 to $w/2$. Clearly, periods larger than $w/2$ are not significant in a data stream window of length w. Therefore, we focus on the values $c_i(S_{0,w})$ for $i = 1, 2, \ldots, w/2$ in the sequence $C(S_{0,w})$. As mentioned earlier, these values correspond to comparing $S_{0,w}$ to its shifted versions $S^{(i)}_{0,w}$ for $i = 1, 2, \ldots, w/2$. Let the window slide $z \leq w$ positions such that the current portion of the data stream is $S_{z,w}$. The SPD algorithm executes over $S_{z,w}$ and the results from the previous and the current portions should be combined. We observe that if $z > w/2$, some comparisons would be missed. For example, if $S = e_0, e_1, \ldots$ and $w = 6$, then $c_3(S_{0,w})$ corresponds to

comparing $S_{0,w} = e_0, e_1, \ldots, e_5$ to $S^{(3)}_{0,w} = ***e_0, e_1, e_2$. For $z = 4$, $c_3(S_{z,w})$ corresponds to comparing $S_{z,w} = e_4, e_5, \ldots, e_9$ to $S^{(3)}_{z,w} = ***e_4, e_5, e_6$. Although comparing e_6 to e_3 should be part of comparing the entire data stream $S = e_0, e_1, \ldots, e_9$ to $S^{(3)} = ***e_0, e_1, \ldots, e_6$, it is not included in either $c_3(S_{0,w})$ or $c_3(S_{z,w})$. Hence, we can deduce that a window of length w should not slide more than $w/2$ positions in order not to lose any comparison information. Moreover, using the same example, we observe that the lower the value of z is, the more overlap we get in symbol comparisons. Consequently, we choose to slide a window of length w a number of positions equal to $w/2$, as illustrated in Figure 1(a).

Recall that the SPD algorithm uses a binary mapping and a modified convolution to perform the shift and compare operations all at once. In such terms, we can define the combined values for the entire data stream, which has arrived so far, as follows:

$$\begin{aligned} &c_i(\bar{S}_{kw/2,w}) = \textstyle\sum_{j=0}^{i} 2^j \bar{e}_{j+kw/2} \cdot \bar{e}_{i-j+kw/2}, \\ &c_i(\bar{S}_{0,w+kw/2}) = 2^{\sigma w/2} c_i(\bar{S}_{0,w+(k-1)w/2}) + c_i(\bar{S}_{kw/2,w}) \end{aligned} \qquad \forall 0 \le i \le \sigma w - 1, \quad \forall k \ge 0$$

At every stage of sliding the window with $w/2$ positions, the previous values are shifted to the left and are added to the new values. Since we deal with powers of 2, shifting $w/2$ positions to the left before the addition operation ensures that every single match has a different power of 2 in the total sum. Hence, we are able to take care of the overlap and discard any overlapped values. Note that in the previous equations, i ranges up to only $\sigma w - 1$ even when the data stream has been extended in length. In other words, only the first w components of the sequence $C(S_{0,w+kw/2})$ are computed.

3.3.2 Multiple Sliding-Windows

After sliding a single window k times, the entire data stream has become of length $w + kw/2$. However, we are still limited to potential period lengths up to $w/2$. Even if we consider period length values larger than $w/2$, we are bounded by w as the maximum period length to be discovered. Thus, the smaller the length of the window, the more potential periods that are missed. On the other hand, the larger the length of the window is, the more time the algorithm waits until it produces interactive output.

Illustrated in Figure 1(b), the solution we propose is to have multiple sliding windows of different expanding lengths that are staggered over the data stream. The computations

are shared among the multiple windows in a way similar to that of sliding a single window. Assume that we have two windows of lengths $w_1 < w_2$, and that all the first w_1 components of the sequence $C(S_{0,w_1})$ are computed. Only the first $w_1/2$ components are of particular interest to discover the potential period lengths up to $w_1/2$. We are now interested in the first w_2 components of the sequence $C(S_{0,w_2})$. We observe that the components that lie between $w_1/2$ and w_2 are the only ones to be computed for the window w_2. The first $w_1/2$ components are updated in the next sliding of the window w_1. Therefore, for the window w_2, we combine the results of the components between $w_1/2$ and w_1, and compute the new components between w_1 and w_2.

$$c_i(\bar{S}_{0,w_2}) = \begin{cases} 2^{\sigma w_1/2} c_i(\bar{S}_{0,w_1}) + \sum_{j=0}^{i} 2^j \bar{e}_j \cdot \bar{e}_{i-j} & \forall \sigma w_1/2 \leq i < \sigma w_1 \\ \sum_{j=0}^{i} 2^j \bar{e}_j \cdot \bar{e}_{i-j} & \forall \sigma w_1 \leq i < \sigma w_2 \end{cases}$$

This process is carried out between each two consecutive windows $w_1 < w_2 < \ldots < w_m$. The selection of w_m is bounded by the buffer size allowed by the system for buffering the data stream. Therefore, $w_m/2$ is the maximum period length value that we can discover.

4 Mining Periodic Patterns

The max-subpattern tree [19] has proved to be efficient in mining periodic patterns in time series data. Han et al. [19] have presented a two-pass algorithm for building such a tree. Aref et al. [7] have presented an incremental version of the algorithm that maintains the max-subpattern tree during continuously arriving data. However, the algorithm in [7] requires two passes over the new data and a possible reprocessing of the previously seen data. In this section, we propose a new incremental technique that fits the data stream model. Data is not stored and hence a reprocessing of the previously seen data is not possible. The produced results are approximate compared to the exact results produced by the two-pass algorithms [19, 7]. Yet, this approximation is unavoidable and is empirically reasonable as the accuracy exceeds 90%.

We give an overview of the max-subpattern tree [19] (Section 4.1). Then, we introduce how STAGGER maintains the max-subpattern tree incrementally over a data stream

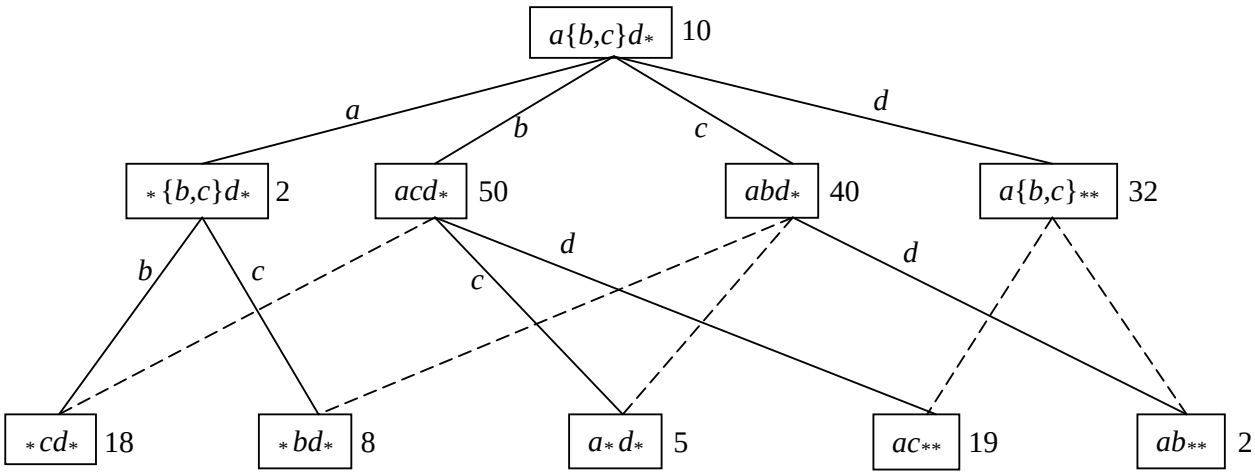


Figure 3: An example of a max-subpattern tree

(Section 4.2). Subsequently, we introduce the "hysteresis" approach for maintaining the periodicity thresholds over the data stream (Section 4.3).

4.1 The Max-Subpattern Tree

Figure 3 gives an example of a max-subpattern tree built with a period of length 4. Every node represents a candidate periodic pattern and has a count that reflects the number of occurrences of this pattern in the data. Recall that, according to Definition 2, a pattern is a sequence of subsets over the entire alphabet. A node is a parent to another if the pattern of the latter node is a subpattern of the pattern of the former node. A pattern q' is called a subpattern of another pattern q if for each position i, $q'_i \subseteq q_i$. The direct link between a parent node and a child node is labeled by the difference between their two patterns. Notice that a pattern may be a subpattern of more than one pattern. Therefore, the data structure is actually a graph rather than a tree. In order to preserve the data structure as a tree and decrease the complexity of insertion and search, not all the parent-child links are kept. The dotted lines in Figure 3 represent those links that are not kept.

Clearly, the root node of the max-subpattern tree should represent the candidate periodic pattern that all the other candidate periodic patterns are subpatterns of. Hence, the root node pattern is called the *maximal* periodic pattern Q. The algorithm of [19] determines Q by extracting all the *single-symbol* patterns of length p from a first pass over the data. The

frequent single-symbol patterns are determined with respect to the periodicity threshold, and Q is computed by the union operation of all these frequent single-symbol patterns. For example, if the frequent single-symbol periodic patterns of length 4 are $*\mathsf{b}**$, $*\mathsf{c}**$, $\mathsf{a}***$, and $**\mathsf{d}*$, then $Q = \mathsf{a}\{\mathsf{b},\mathsf{c}\}\mathsf{d}*$.

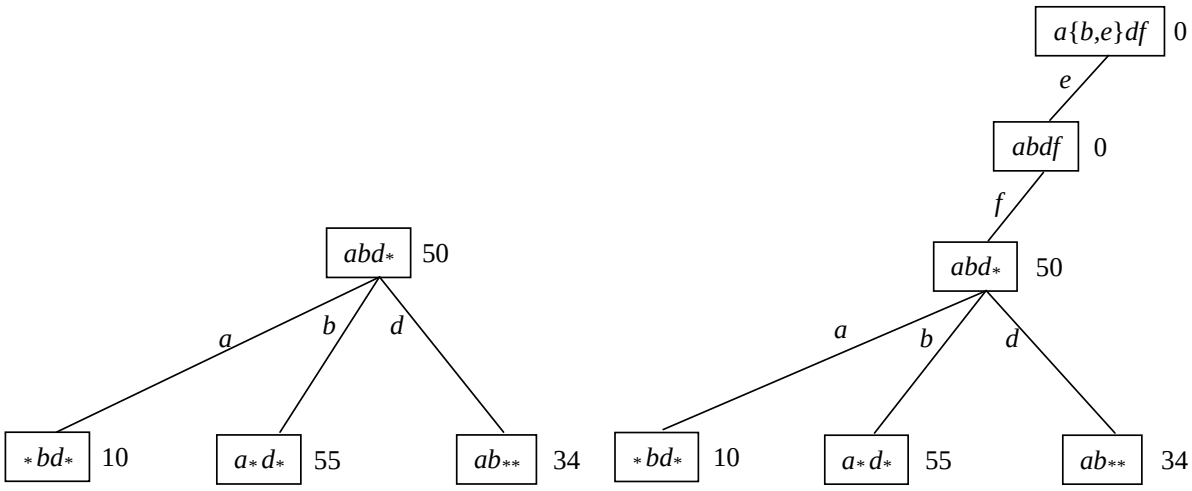
A second pass over the data combines every p symbols and inserts the formed pattern, filtered by the maximal periodic pattern, into the max-subpattern tree. Once it is built, the max-subpattern tree is traversed in order to detect those actual frequent periodic patterns with respect to the periodicity threshold. Note that the count of every pattern is actually the count of its corresponding node plus all the counts of its parent nodes (both the direct parent and the other hidden parents).

Note that in STAGGER, that first pass is not conducted. The periodicity detection technique (Section 3) reveals not only the potential period lengths but also their single-symbol periodic patterns that form their corresponding maximal periodic patterns.

4.2 Approximate Incremental Technique

The idea of our proposed technique for mining periodic patterns in data streams is to maintain the max-subpattern tree over the continuous arrival of new data. The arrival of new data may update the maximal periodic pattern Q due to the discovered single-symbol periodic patterns. Accordingly, the max-subpattern tree should be updated. Let Q and Q' be the maximal periodic patterns before and after the update, respectively. Let c_j and c'_j be the components at position j of Q and Q', respectively. If $c'_j \neq c_j$, then updating c_j to c'_j implies deletion and/or insertion of one or more symbols. For example, if $Q = \mathsf{a}\{\mathsf{b},\mathsf{c}\}\mathsf{d}*$ and $Q' = \mathsf{a}\{\mathsf{b},\mathsf{e}\}\mathsf{df}$, then updating Q to Q' implies deleting the symbol c from position 2, and inserting the symbols e and f at positions 2 and 4, respectively.

STAGGER updates the max-subpattern tree in two steps: a deletion step followed by an insertion step. Let Q^t be the pattern resulting from deleting the necessary symbols from Q, e.g., in the previous example, deleting the symbol c from position 2 results in $Q^t = \mathsf{abd}*$. If the original max-subpattern tree R contains a node representing the pattern Q^t, then STAGGER converts that node to be the root of the updated max-subpattern tree. Otherwise, STAGGER creates a new root node representing the pattern Q^t. There are two



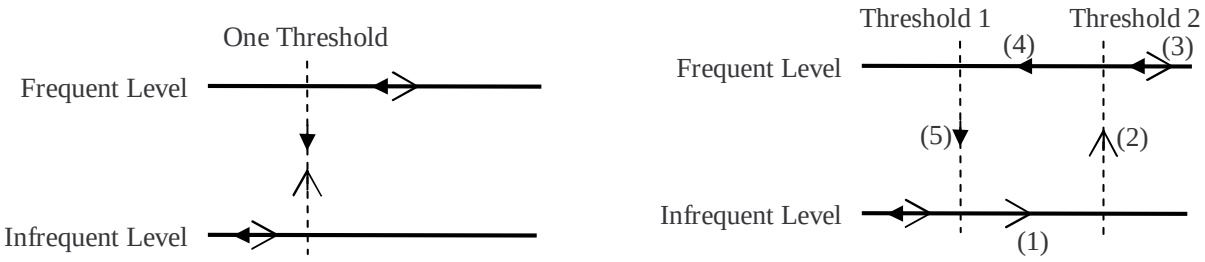
(a) After deleting the symbol c

(b) After inserting the two symbols e and f

Figure 4: Updating the max-subpattern tree of Figure 3

aspects concerning that intermediate resulting tree. First, the counts must be fixed since some deleted nodes were parents to some retained nodes (e.g., in Figure 3, deleting the root node means a missing count of 10 for all its children nodes). Second, the non-linked children from R should be added. To consider both of these two aspects simultaneously, STAGGER scans the original max-subpattern tree R and inserts R's patterns into the intermediate max-subpattern tree with their corresponding counts. For example, in Figure 3, where $Q = \mathsf{a\{b,c\}d*}$, assume that $Q' = \mathsf{a\{b,e\}f}$, then $Q^t = \mathsf{abd*}$. The intermediate max-subpattern tree is given in Figure 4(a). Notice that, without fixation, the intermediate max-subpattern tree would have had only two nodes: one for the pattern $\mathsf{abd*}$ with a count of 40, and one for the pattern $\mathsf{ab**}$ with a count of 2.

In the insertion step, STAGGER creates a new root node for the updated max-subpattern tree that represents the updated maximal periodic pattern Q'. Nevertheless, the updated max-subpattern tree is not accurate since it has no information about those inserted symbols. In other words, the counts of the newly created nodes along the path from the new root to the old root are set to 0. The only way to have their real counts is to reprocess the previous data, which is not possible. This is why STAGGER is considered an approximate algorithm. For example, Figure 4(b) shows the max-subpattern tree updated from Figure 4(a) to have $Q' = \mathsf{a\{b,e\}df}$ as its root node. Although the previously seen data might contain patterns



(a) Traditional thresholding approach

(b) Hysteresis thresholding approach

The big wide arrows show the path of an infrequent-becoming-frequent pattern, and the small solid arrows show the path of a frequent-becoming-infrequent pattern.

Figure 5: Hysteresis versus traditional thresholding approaches

that had the symbol f in position 4, the fact that this symbol was not frequent before had prohibited us from including any pattern containing this symbol in the max-subpattern tree.

Now that the max-subpattern tree is updated to reflect the change of the maximal periodic pattern, the newly arrived data patterns are inserted into the updated tree. Thus, the max-subpattern tree now represents approximately the whole data stream.

4.3 The "Hysteresis" Thresholding Approach

The proposed technique, as well as any traditional data mining technique, handles the patterns as follows. As long as a pattern, say q, is not frequent, q is not included in the max-subpattern tree. As soon as q becomes frequent, it is added to the max-subpattern tree, and its history starts. As long as q is frequent, it is kept in the max-subpattern tree updating its history. As soon as q becomes infrequent, it is removed from the max-subpattern tree losing its history information. The frequency of the pattern is determined by a single threshold. This approach is illustrated in Figure 5(a). A disadvantage of this approach is that it fails to handle a pattern that oscillates between being frequent and being infrequent. Such a pattern, say q, will lose all the history information (max-subpattern tree counts) as soon as it becomes infrequent. When q becomes frequent again, it will be treated as a newly appeared frequent pattern which has no history information.

One can think of several approaches for overcoming that disadvantage. One of which is to not remove an infrequent pattern from the max-subpattern tree as it may become frequent later. This approach, however, suffers the disadvantage that the max-subpattern tree will be

huge and will contain several infrequent patterns, especially when a data stream has a rather lengthy transient component. Another approach is to have another max-subpattern tree for those patterns that are infrequent but are candidates to be frequent later. Those patterns are determined by a lower threshold value than the original one. Yet, this approach migrates the problem to that other max-subpattern tree, which will suffer the same disadvantage with a lower threshold value.

We propose a tradeoff approach whose idea comes from the Physics domain. Hysteresis represents the *history* dependence of physical systems[6]. The proposed approach utilizes two thresholds, as illustrated in Figure 5(b). A pattern is considered frequent if its frequency is above the higher threshold value, and is considered infrequent if its frequency is below the lower threshold value. The distance between the two threshold values acts as a period of time during which a pattern is given a chance to stabilize rather than to oscillate.

5 Experimental Study

This section contains the results of an experimental study that examines STAGGER for different aspects. In Section 5.1, experiments are conducted using synthetic data in order to examine the accuracy and the time performance of STAGGER. In Section 5.2, the practicality and usefulness of the output are explored using real data experiments.

We generate controlled synthetic data by tuning some parameters, namely, data distribution, period length, alphabet size, and noise amount. Both uniform and normal data distributions are considered. First, inerrant data is generated by repeating a partial pattern that is randomly generated from the specified data distribution. Then, noise is introduced randomly and uniformly over the whole data stream. Noise is introduced by altering a symbol by another, inserting a new symbol, or deleting the current symbol at a randomly selected position in the data stream. Unless stated otherwise, data streams lengths of 1M symbols are used with alphabet size of 10, and the values collected are averaged over 100 runs. With respect to the period lengths, we consider values that divide the whole data stream length as

[6]If you push on something, it will yield, and when you release, if it does not spring back completely, then it is exhibiting hysteresis.

well as values that do not. We thus inspect STAGGER bias, if any, towards any particular set of period length values.

STAGGER is the first one-pass online incremental periodicity mining algorithm for data streams that combines both periodicity detection and periodic patterns mining. Hence, there is no direct comparison between STAGGER and any of the algorithms in the literature. However, we compare our periodicity detection technique to the periodicity detection algorithm, AWSOM [29]. AWSOM uses Wavelets to approximate data streams and hence discovers potential periodicity rates.

5.1 Synthetic Data Experiments

The first experiment studies the accuracy of the proposed incremental technique for mining periodic patterns. The fact that STAGGER is approximate in building the max-subpattern tree in one pass implies how the accuracy is measured. The approximate tree is compared to the accurate tree that is built offline in a two-pass algorithm [19]. The relative error of each node in the tree is computed and averaged over the total number of nodes in the tree. In Figure 6, we use the symbols "U" and "N" to denote the uniform and the normal distributions, respectively; and the symbol "P" to denote the embedded period length.

When a single sliding window is used, Figure 6(a) shows expectedly that increasing the window length increases the accuracy since the effect of transient frequent periodic patterns is reduced. Furthermore, as the stream data continuously arrives, the accuracy increases, as shown in Figure 6(b), since the data becomes progressively more stable. Therefore, the frequent periodic patterns are determined and the probability of adding new frequent periodic patterns is very low. Both Figures 6(a) and 6(b) show that STAGGER is highly accurate (in the 90% level).

Figure 6(c) gives the results of inspecting the "hysteresis" thresholding approach by varying the distance between the two threshold values. A zero distance means only one threshold value, that is the traditional thresholding approach. Figure 6(c) shows that the "hysteresis" thresholding approach increases the accuracy of the approximate max-subpattern tree. As the distance between the two threshold values increases, the accuracy increases since we allow more time for a pattern not to lose its history (max-subpattern tree counts). After

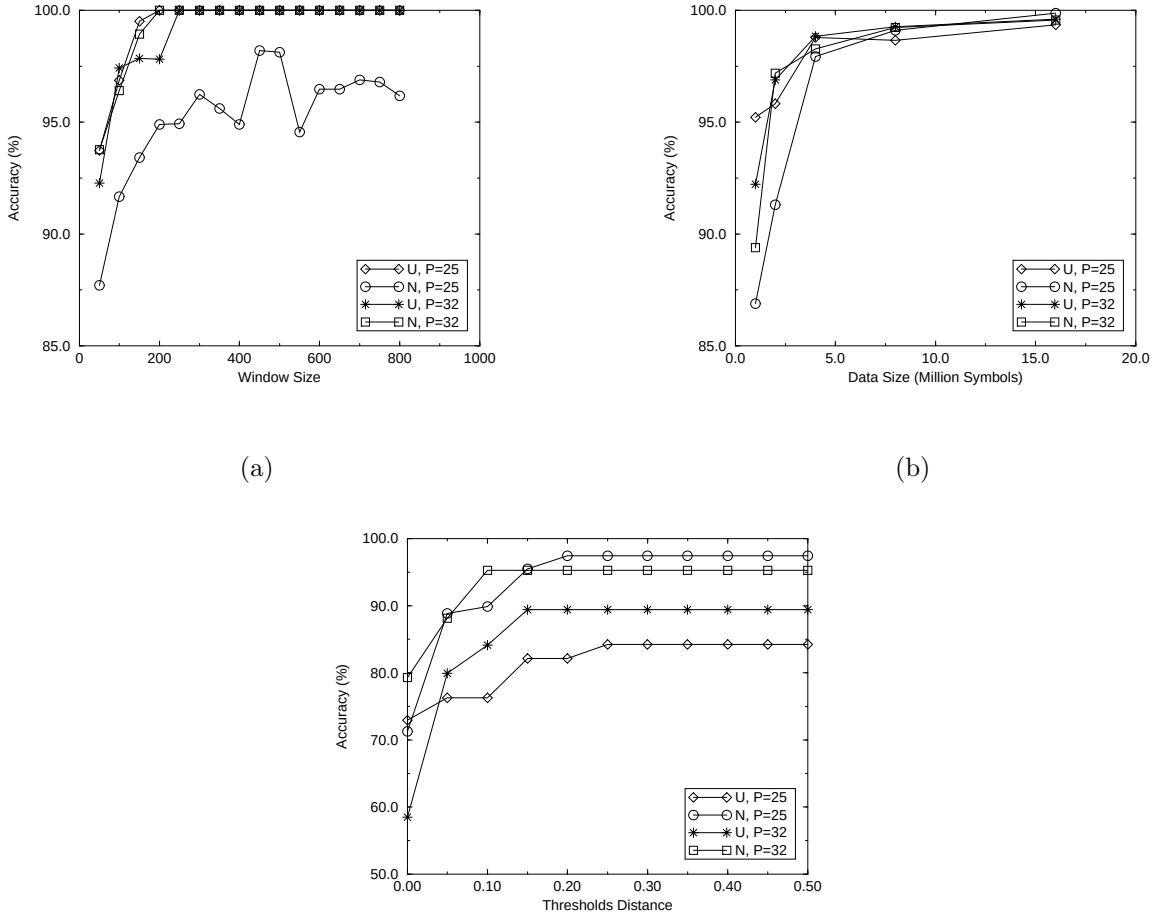


Figure 6: Accuracy of the approximate max-subpattern tree

some distance level, the accuracy does not increase more since the allowed time is more than enough for all patterns to keep their history.

The next experiment inspects the accuracy of STAGGER with respect to the discovered potential period lengths. The accuracy measure that we use is the ability of STAGGER to detect all the period lengths that were artificially embedded into the synthetic data. To discover a period length accurately, it is not enough to discover it at any periodicity threshold value. In other words, the period lengths discovered with a high periodicity threshold value are better candidates than those discovered with a lower periodicity threshold value. Therefore, we define the confidence of a discovered period length to be the minimum periodicity threshold value required to detect this period length. The accuracy is measured by the

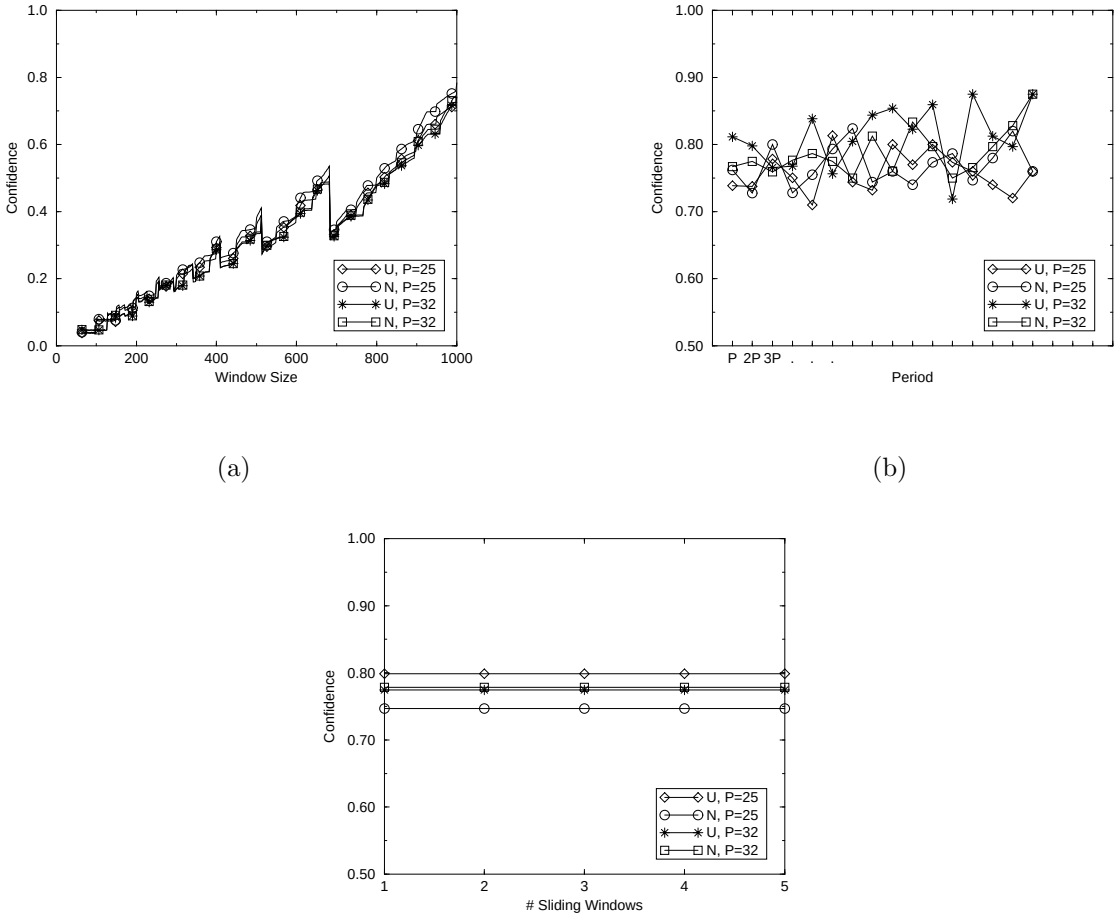
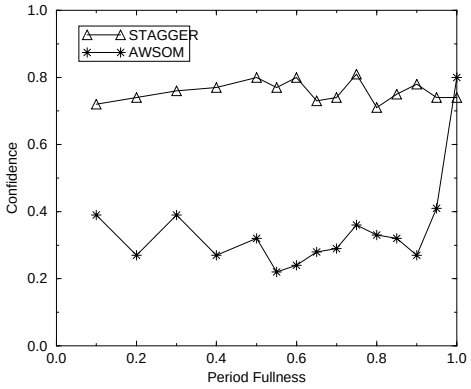


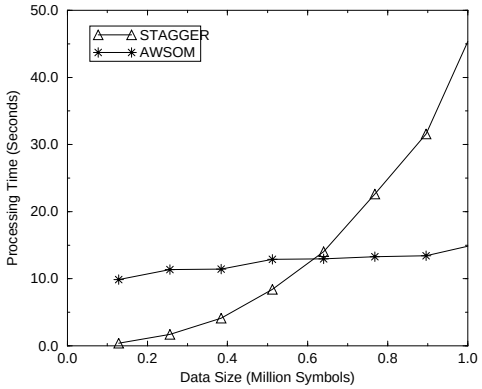
Figure 7: Accuracy of symbol periodicity detection

average confidence of all the period lengths that are embedded artificially into the synthetic data. Figure 7 gives the results of this experiment. Recall that synthetic data is generated such that the embedded period lengths are: $P, 2P, \ldots$.

Using a single sliding window, Figure 7(a) shows that increasing the window length increases the accuracy since this allows the algorithm to detect more period lengths (of larger lengths). Subsequently, using a single sliding window of maximum buffer size, Figure 7(b) shows an unbiased behavior with respect to the embedded period length. Moreover, Figure 7(c) shows that the accuracy does not rely on the number of expanding sliding windows as far as the largest sliding window is selected to be as large as the buffer size in order to detect all possible period lengths. Both Figures 7(b) and 7(c) show that STAGGER achieves



(a) Accuracy



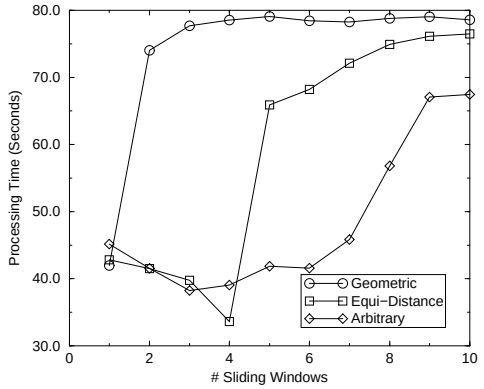
(b) Time Performance

Figure 8: STAGGER versus AWSOM

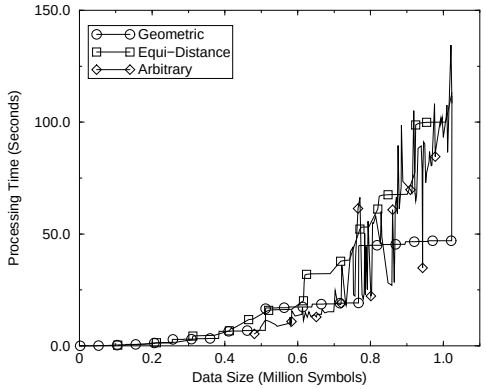
an average of 80% accuracy, i.e., STAGGER detects all the artificially embedded period lengths at a periodicity threshold value of 80%.

To compare STAGGER to the periodicity detection algorithm of [29] (AWSOM), we consider both the time performance and the accuracy of the discovered potential periodicity rates. AWSOM has two main limiting features. First, AWSOM discovers only periodicity rates of lengths that are powers of two. This limitation is due to the use of Wavelets to approximate the data stream. Periodicity rates of lengths that are not powers of two will be missed by AWSOM. The second limitation of AWSOM is that it discovers periodicities that are complete, e.g., full sinusoidal periods. If a period has only few periodic symbols, AWSOM fails to discover such a period. On the other hand, STAGGER does not suffer either disadvantage as (i) it uses convolution to consider all possible period lengths, and (ii) it defines the periodicity in terms of its symbols. Figure 8(a) shows empirically this latter property. The experiment compares STAGGER to AWSOM based on their ability to discover semi-full periods. The experiment considers only periodicity rates of powers of two. Figure 8(a) shows that both algorithms achieve 80% accuracy at a full period (*1.0 period fullness*). However, for semi-full periods, STAGGER outperforms AWSOM with an order of magnitude.

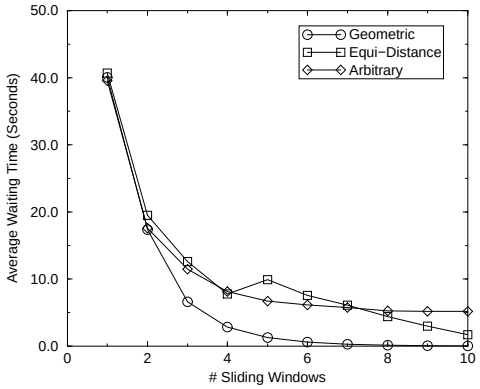
On the other hand, Figure 8(b) shows that AWSOM outperforms STAGGER with respect to the processing time. AWSOM is a Wavelet-based algorithm that takes $O(n)$ time to



(a)



(b)



(c)

Figure 9: Time performance of STAGGER

compute. However, STAGGER takes $O(n \log n)$ time to compute the convolution. In conclusion, STAGGER trades processing time for more accurate periodicity rates, more general periodicities in terms of symbols, and more discovered information (the periodic patterns themselves). In contrast, AWSOM trades accuracy for processing time by the Wavelet approximation that may lead to discarding potential periodicities.

To evaluate the time performance of STAGGER, we observe that the number of expanding sliding windows is the key factor of the processing time, as depicted in Figure 9(a). We have three different settings for the lengths of the expanding sliding windows: arbitrary, geometric. and equi-distance settings. Let m be the number of expanding sliding windows of lengths $w_1 < w_2 < \ldots < w_m$ such that w_m is selected to be as large as the buffer size. For

all $i = 1, 2, \ldots, m-1$, while the arbitrary setting selects w_i randomly, the geometric setting selects w_i to equal $w_{i+1}/2$, and the equi-distance setting selects w_i to equal $(i/m)w_m$. Figure 9(a) shows that the arbitrary setting gives the least processing time while the geometric setting gives the most processing time.

Figure 9(b) shows that the processing time of both the geometric and the equi-distance settings increases gradually over time (as the data stream increases in size), whereas the processing time of the arbitrary setting increases irregularly. The importance of the gradual increase is that it gives a bound on the data arrival rate that STAGGER can manage without dropping any data symbols. For example, Figure 9(b), drawn with 10 expanding sliding windows, shows that STAGGER with the geometric setting has processed 1M symbols in approximately 100 seconds. This means that STAGGER can cope with an arrival rate of 10,000 symbols per second. Of course, this high rate is due to the high speed processor we used in our experiments.

The most important advantage of having expanding sliding windows is shown in Figure 9(c), that is a much better average waiting time over a single sliding window. Clearly, the more the number of expanding sliding windows, the less average waiting time between consecutive outputs. Moreover, Figure 9(c) shows that the geometric setting of the expanding sliding windows has the best performance with respect to the average waiting time.

5.2 Real Data Experiments

A real-world database that contains sanitized data of timed sales transactions for some stores over a period of 15 months serves the purpose of real data experiments. Timed sales transactions data, of size 130 Megabytes is streamed to STAGGER. Although this data stream has a slow arrival rate (one value per hour), the purpose of the experiment is to demonstrate the practicality of the discovered periodic patterns. The numeric data values are discretized into five levels, i.e., the alphabet is of size 5. The levels are *very low*, *low*, *medium*, *high*, and *very high*. Discretization is based on manual inspection of the values (*very low* corresponds to zero transactions per hour, *low* corresponds to less than 200 transactions per hour, and each level has a 200 transactions range). Notice that although data is in terms of transactions per hour, the algorithm streams the data at a much higher rate.

Periodicity Threshold (%)	# Output Periods	Some Output Periods
50	3164	263, 409
55	2777	337, 385
60	2728	481
65	2612	503
70	2460	505
75	2447	577, 3961
80	2328	647
85	2289	791
90	2285	721
95	2281	24, 168

Table 1: Symbol periodicity detection output

Periodic Pattern	Frequency(%)
aaaa∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗aaa	49.78166
aaaa∗∗∗b∗∗∗∗∗∗∗∗∗∗∗∗aaa	42.57642
aaaa∗∗∗b∗∗∗∗∗∗∗∗d∗∗∗aaa	38.56768
∗∗∗∗∗bbbbc∗∗∗∗∗∗∗∗∗∗∗∗aa	35.80786

Table 2: Mining periodic patterns output

Table 1 gives the output of STAGGER symbol periodicity detection for the real data for different values of the periodicity threshold. Clearly, STAGGER outputs fewer period lengths for higher threshold values, and the period lengths detected with respect to a certain value of the threshold are enclosed within those detected with respect to a lower value. To verify its accuracy, STAGGER should at least output the period lengths that are expected in the data. Our real data has an expected period of length 24 that corresponds to the daily pattern of number of transactions per hour. Table 1 shows that a period of length 24 is detected when the threshold is 95% or less. In addition, STAGGER detects many more period lengths, some of which are quite interesting. A period of length 168 (24×7) can be explained as the weekly pattern of number of transactions per hour. A period of length 3961 shows a periodicity of exactly 5.5 months plus one hour, which can be explained as the daylight savings hour. One may argue against the clarity of this explanation, yet this proves that there may be obscure periods, unknown a priori, that STAGGER can detect.

Analyzing the output for the period of length 24, there are 13 symbol periodicities (single-symbol periodic patterns) with respect to a periodicity threshold of 50%. The pat-

tern aaaa ∗ bbbbc ∗ ∗ ∗ ∗ ∗ ∗ ∗ d ∗ ∗ ∗ aaa is discovered to be the maximal periodic pattern of length 24. Table 2 gives the final output of STAGGER only for the period of length 24. Knowing the meaning of every symbol leads to useful interpretation of the patterns. For example, the first pattern enunciates that on about 50% of the days, the number of transactions that occur at the first 4 hours and the last 3 hours of the day (9:00PM – 4:00AM) is equal to zero. This periodic pattern, and alike ones, can help the store manager to decide when to close the store, or when to reduce the number of sales assistants.

6 Related Work

Related work falls into two main categories: data mining in data streams, and time series data mining.

Decision trees and clustering are the first two data mining techniques to be investigated in data streams. Domingos et al. [13, 21] have presented an incremental technique for building decision trees over high-speed data streams. Guha et al. [18, 27] have extended the k-median clustering algorithm to fit the data stream model. In [1, 2], Aggarwal et al. have proposed a more general framework for clustering and classifying data streams taking into consideration the evolution nature of data streams. Another data mining technique that has been addressed in the data stream context is regression analysis [10] that has been used for summarizing data streams and building online data cubes. Similarity search queries for patterns have been addressed recently in [16] using Fourier transform. As a related issue to most data mining techniques, counting the data items and finding the most frequent ones have been studied for data streams in [9, 26].

Research in time series data mining has concentrated on discovering different types of patterns. Agrawal et al. [3, 4] have developed a model for similarity of time sequences that can be used for mining periodic patterns. In [6, 30], Agrawal and Srikant have presented Apriori-like [5] techniques for mining sequential patterns, which has been studied further by Garofalakis et al. [17] and by Ayres et al. [8]. Ozden et al. [28] have studied mining of association rules of periodic nature. Han et al. [20, 19] have introduced the notion of partial periodic patterns and have presented two algorithms for mining this type of patterns. That

work has been extended by Yang et al. [32] to account for the intervention of random noise, and by Ma and Hellerstein [25] to account for the case when the period is unknown a priori.

Fourier transform is a basic approach for discovering periodicities of signals. It has the advantages that it is computationally efficient as it requires $O(n \log n)$ time for a signal of length n using FFT [23], and is scalable for large values of n as it can operate externally (on disk) using external FFT [31]. However, it does not discover periodicities with respect to symbols, and deals with fixed length signals and cannot be updated as new points arrive.

7 Conclusions

In this paper, we have proposed an online and incremental algorithm, named STAGGER, for periodicity mining in data streams. STAGGER is novel in the sense that it detects periodicity rates and discovers frequent periodic patterns in one pass through the data. This feature is essential when dealing with potentially infinite streams. To maximize the coverage over the stream, STAGGER uses expanding sliding windows to detect all periodicity rates. STAGGER maintains a tree-like data structure to discover the frequent periodic patterns. We have proposed a new approach for handling the thresholds in order to improve the accuracy of STAGGER. An empirical study using synthetic and real data validates the accuracy of STAGGER, shows the usefulness of the output, and shows the tradeoff between accuracy (favoring STAGGER) and processing time (favoring Wavelet-based techniques).

References

[1] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, September 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P. Yu. On demand classification of data streams. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining*, Seattle, Washington, August 2004.

[3] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time series databases. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[4] R. Agrawal, G. Psaila, E. Wimmers, and M. Zait. Querying shapes of histories. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, September 1994.

[6] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[7] W. Aref, M. Elfeky, and A. Elmagarmid. Incremental, online, and merge mining of partial periodic patterns in time-series databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(3):332–342, 2004.

[8] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 2002.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium of Automata, Languages and Programming*, Malaga, Spain, July 2002.

[10] Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

[11] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[12] C. Daw, C. Finney, and E. Tracy. A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, 74(2):915–930, 2003.

[13] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, Massachusetts, August 2000.

[14] M. Elfeky, W. Aref, and A. Elmagarmid. Using convolution to mine obscure periodic patterns in one pass. In *Proceedings of the 9th International Conference on Extending Data Base Technology*, Heraklion, Crete, Greece, March 2004.

[15] M. Elfeky, W. Aref, and A. Elmagarmid. Periodicity detection in time series databases. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[16] L. Gao and X. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, Wisconsin, June 2002.

[17] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, United Kingdom, September 1999.

[18] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, California, November 2000.

[19] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series databases. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, March 1999.

[20] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time related databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, New York City, New York, August 1998.

[21] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, August 2001.

[22] E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. In M. Last, A. Kandel, and H. Bunke, editors, *Data Mining in Time Series Databases*. World Scientific Publishing, June 2004.

[23] D. Knuth. *The Art of Computer Programming*, volume 2 of *Series in Computer Science and Information Processing*. Addison-Wesley, Reading, Massachusetts, second edition, 1981.

[24] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, San Diego, California, June 2003.

[25] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, April 2001.

[26] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, August 2002.

[27] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, California, February 2002.

[28] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the 14th International Conference on Data Engineering*, Orlando, Florida, February 1998.

[29] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, September 2003.

[30] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Data Base Technology*, Avignon, France, March 1996.

[31] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, June 2001.

[32] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, Massachusetts, August 2000.